Building a Framework to Encourage the use of Metadata in Modern Web-Design

Poster

Jackson Morgan
Georgia Institute of Technology, United States of America
jmorgan45@gatech.edu

Keywords: javascript, event-driven, framework, web development, rdf, sparql

Abstract

When Tim Berners-Lee published the roadmap for the semantic web in 1998, it was a promising glimpse into what could be accomplished with a standardized metadata system, but nearly 20 years later, adoption of the semantic web has been less than stellar. In those years, web technology has changed drastically, and techniques for implementing semantic web compliant sites have become relatively inaccessible. This poster outlines a JavaScript framework called Beltline.js which seeks to encourage the use of metadata by making it easy to integrate into modern web best-practices.

Introduction

As interactive websites have become more ubiquitous, JavaScript has increasingly been a must-know language for web developers. It has become a household name for developing applications becoming by far the most used language on github making up 15% of the site at over 300,000 repositories (Zapponi, 2014).

Because of the popularity of JavaScript, libraries targeted at JavaScript developers have been able to introduce novel concepts to a wide audience. One such library that would be of interest to the semantic web community is Facebook's GraphQL (GraphQL, 2018). GraphQL provides a query language that allows a client to define the structure of information it wants to receive before querying the server. In a way, it's reminiscent of a few features in the Resource Description Framework (RDF) and its query language SPARQL. However, GraphQL is not designed to further the availability of metadata. Unlike RDF which is designed to link many domains together with triples, GraphQL is built around the traditional idea of isolated servers providing domain specific data. Nonetheless, because GraphQL is a JavaScript framework, a new generation of developers was introduced to the concept of graph-based data.

Beltline.js is a JavaScript library that aims to capitalize on the viral nature of JavaScript libraries while increasing the adoption of RDF. Named after the Beltline in Atlanta Georgia, an ambitious path and transit corridor that will connect a plethora of neighborhoods in a shared travel experience, Beltline aims to connect the various aspects of a JavaScript web application through a shared data interface.

Architectural Inspiration

Event-driven architectures have become increasingly popular among web developers. By utilizing multi-directional communication, oftentimes implemented with WebSockets, a developer is able to build an experience that keeps a user interface up-to-date with the application as a whole. Almost all interfaces that push updates to users without requiring a user to reload, navigate to a new page, or interact with the UI are supported event-driven architectures. (Michelson, 2011)

Beltline is heavily influenced by the Distributed Data Protocol (DDP), a standard for event driven architectures most known for its use in the popular JavaScript framework, Meteor.js. (DDP, 2016) While originally designed to integrate with the database MongoDB, with a few modifications, DDP can work with triplestore databases. Beltline's implementation is designed to accomplish 3 architectural goals in order to achieve developer ease:

Event-Driven: It should not rely on a request-reply architecture

Controllable: A developer should be able to easily control how much data a client is given

Integrable: It should be easy for a developer to integrate Beltline into their current tech-stack.

Implementation

Beltline's *event-driven* architecture is enabled by WebSockets. When a web browser connects to a Beltline-enabled web site, it makes a WebSocket (WebSockets, 2018) connection with the server using Socket.io (Socket.io, 2018). This enables not only a client's ability to push data to the server but the server's ability to push data directly to its clients unprompted.

Beltline's method of syncing data between the client and the server takes heavy inspirations from Meteor's Mini-Mongo (MiniMongo, 2018), but using the JavaScript triplestore, rdfstore-js (rdfstore-js, 2018), a JavaScript library that has the same interface and functionality as a normal triplestore database. As it is completely built in JavaScript, our JavaScript triplestore can be instantiated inside a web page upon load. This allows a user to query a database locally as if the database were on the same machine. Beltline keeps each instance of the JavaScript triplestore across all connected web pages in sync with the main triplestore database on a developer's server. When a request is made to update the database using Beltline's call method, Beltline follows the optimistic UI design pattern (Stubailo, 2015). The request first updates the JavaScript triplestore as if nothing went wrong. It then makes the request to the server, and when the request is properly received, it updates all other clients with the new information.

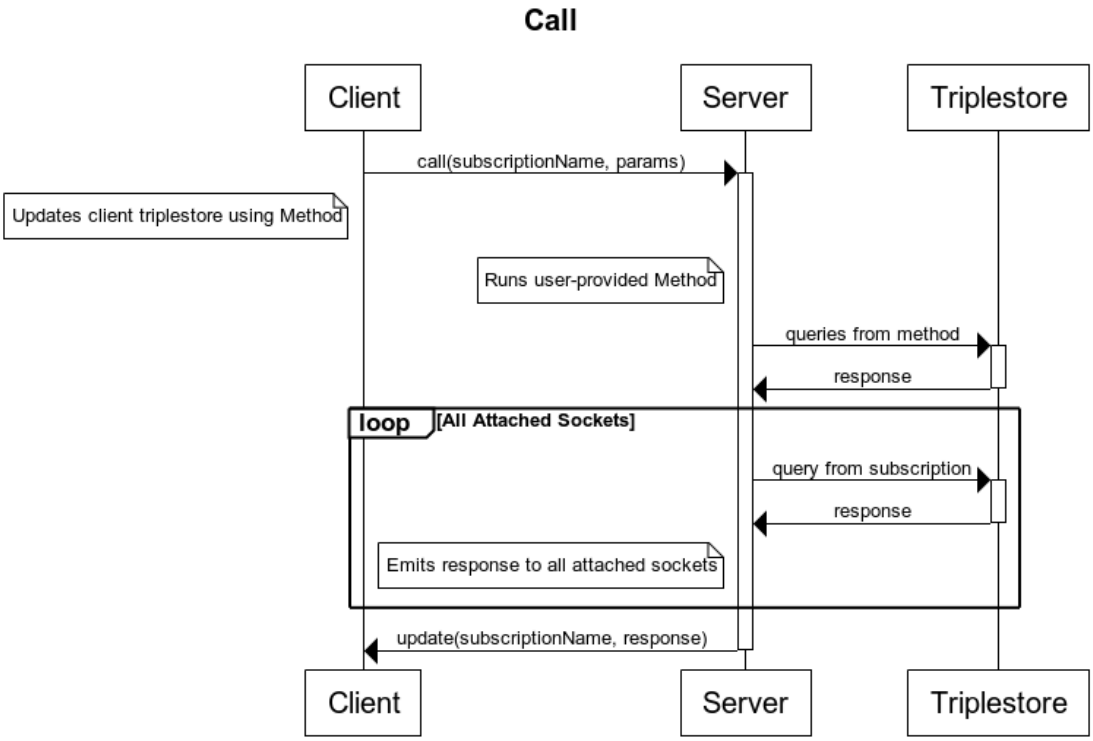

FIG. 1. Sequence for making a database update.

One major downfall of replicating a database across many clients is that it is not *controllable*. There are very few applications that would be designed to load the entirety of a database into a web page. Fortunately, DDP offers a solution to this conundrum in the form of the publish and subscribe methods. Beltline follows suit, employing the same functionality and method names.

Beltline's publish method lives on the server and accepts an id to denote what is being published by taking a user-defined function as parameters. On the client, the subscribe method can be called

by passing in an id corresponding to one of the publish functions. At this point, the function will run a SPARQL query on the server, extracting the desired data. That data is then sent as triples to the client to be stored in the client's JavaScript triplestore. It should be noted that these methods only work with CONSTRUCT SPARQL queries as the result must be a graph to be shared between the server and client.

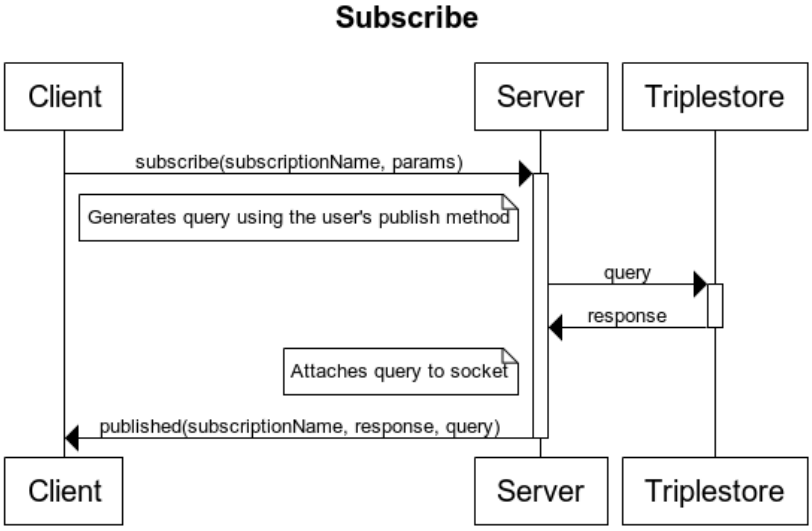

FIG. 2. Sequence for subscribing and publishing.

Finally, in order to increase adoption, Beltline must be *integrable* in frameworks with which developers are familiar. To integrate Beltline, a developer simply needs to import Beltline's server middleware, configure it, and attach it to a route using Express (a popular JavaScript server framework) or some other server framework. Configuration options include the IP address of the main database and optional database plugins. Plugins serve the purpose of extending the usability of Beltline to developers who do not use a SPARQL compatible database. For example, one plugin could transform Beltline's SPARQL queries to SQL queries to hook into a developer's SQL database (Prud'hommeaux, 2009).

Because there is a wide array of client-side JavaScript frameworks, Beltline's frontend libraries must be framework agnostic. As a result, Beltline relies on callback functions that can be passed into the subscribe or call methods at any point in the code. When an event happens, these functions will be triggered, which could lead to an update anywhere else in the client-side codebase.

Beltline not only provides a convenient solution for developers hoping to build event-driven web applications, it also encourages them to open a SPARQL endpoint for their data. Often, making a site semantic web compliant as an extraneous task for the developer, and her effort could be better spent developing new features. Beltline makes feature development and the exposure of semantic data one in the same by making metadata core to a JavaScript framework.

A demo of Beltline can be viewed at https://github.com/jaxoncreed/beltline-example and the full implementation can be installed at https://www.npmjs.com/package/beltline.

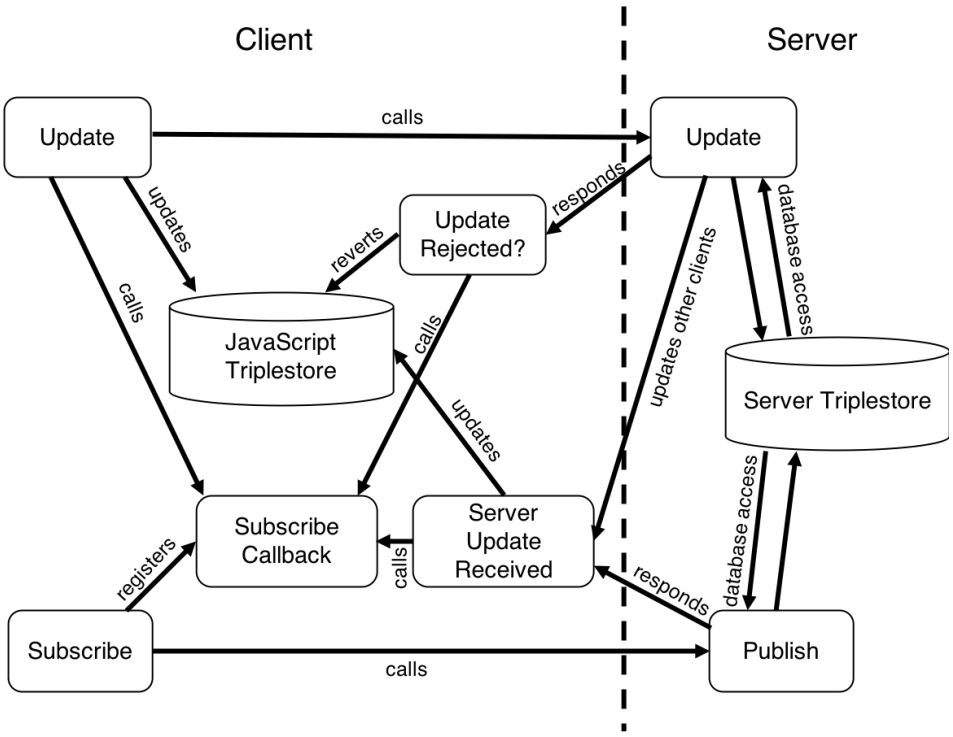

FIG. 3. Beltline Data-Flow.

Conclusion

Beltline not only provides a convenient solution for developers hoping to build event-driven web applications, it also encourages them to open a SPARQL endpoint for their data. Often, making a site semantic web compliant as an extraneous task for the developer, and her effort could be better spent developing new features. Beltline makes feature development and the exposure of semantic data one in the same by making metadata core to a JavaScript framework.

References

DDP. Retrieved, August 16, 2018, from https://github.com/meteor/meteor/blob/devel/packages/ddp/DDP.md.

GraphQL. Retrieved, May 6, 2018, from https://graphql.org.

Meteor. Retrieved, May 6, 2018, from https://www.meteor.com.

Michelson, Brenda M. (2011, February). Event-Driven Architecture Overview. Elemental Links. Retrieved, May 6, 2018, from http://elementallinks.com/el-reports/EventDrivenArchitectureOverview_ElementalLinks_Feb2011.pdf.

MiniMongo. Retrieved, May 6, 2018, from https://github.com/mWater/minimongo.

Prud'hommeaux, Eric, and Alexandre Bertails. (2009). A mapping of sparql onto conventional sql. World Wide Web Consortium (W3C). Retrieved, May 6, 2018, from https://www.w3.org/2008/07/MappingRules/StemMapping.

rdfstore-js. Retrieved, May 6, 2018, from https://github.com/antoniogarrote/rdfstore-js.

Socket.io. Retrieved, May 6, 2018, from https://socket.io.

Stubailo, Sashko. (2015, May). Optimistic UI with Meteor. Retrieved, May 6, 2018, from https://blog.meteor.com/optimistic-ui-with-meteor-67b5a78c3fcf.

WebSockets. (2018, March). Mozilla Developer Network. Retrieved, May 6, 2018, from https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

Zapponi, Carlo. (2014). GitHut – Programming Languages and Github. Retrieved, May 6, 2018, from http://githut.info